

Sprint 3:

Le troisième sprint de notre consiste à créer une interface mobile or web qui contient un petit menu des les catégories des articles , et lorsque une click sur un des catégories comme sport par exemple , des articles liés au sport seront affichés.

Problématique

pour réussir ce sprint nous devons accomplir les cibles suivants .

- choisir le type d'interface et l'a développé.
- créer une base de données avec django.
- améliorer notre code de scraping (ça nous aidera pour obtenir plus de data).
- stocker les données obtenue dans la db .
- liée l'application avec le script pour faire le display .

1-choisir le type d'interface et l'a développé

Pour la création de l'interface , nous avons opté pour l'utilisation de l'android ce qui signifie que nous avons choisis le développement web

Android studio

Android Studio est l'environnement de développement intégré (IDE) officiel pour développer des applications Android. Il est basé sur l'IDE de Java IntelliJ IDEA, qui est développé par JetBrains.

Avant Android Studio, de 2009 à 2014, Google proposait comme IDE une distribution spécifique de Eclipse, appelée ADT, avec des plugins contenant notamment le SDK de Android [1]. Android Studio a été annoncé par Google le 15 mai 2013 avec une version bêta, qui est conçu spécifiquement pour le développement Android et ses besoins. Le 8 décembre 2014, Android Studio passe à version stable 1.0 et L'IDE devient alors officiel par Google, et ADT est délaissé. La version actuelle d'Android Studio est la 3.1.4 sortie en août 2018 [2]. Cette version propose des améliorations pour simplifier la vie des développeurs et permet également de supporter la version d'Android Pie (API 28). Comme seul inconvénient, Android Studio est plus gourmand que ADT en consommant beaucoup de mémoire (généralement plus d'1 Go de RAM), même s'il s'améliore au fils des versions.



Android Studio regroupe la richesse de IntelliJ IDEA combinée avec les capacités qui sont nécessaires pour le développement sous Android. Il permet de mettre rapidement en place de nouveaux projets sur le système Android, pour créer une application d'interface utilisateur, ajouter des composants basés sur l'API Android, déboguer vos applications en utilisant les outils du SDK de Android, et même d'exécuter vos applications sur un appareil mobile ou sur un émulateur. Android Studio intègre aussi le moteur Gradle, qui automatise les tâches et permet un développement simplifié. Il nécessite également l'installation du Java Runtime Environment (JRE) pour que toutes ses fonctionnalités soient opérationnelles.

Création d'un nouveau projet Android

Dans Android Studio, pour créer un nouveau projet Android, il faut :

- 1) Cliquer sur : File → New → New Project...
- 2) Remplir les champs concernant le nom de l'application :

Application Name : LearnApp

Compagny Domain : iam.sci, ce qui donnera comme nom de package **sci.iam.learnapp**

Project location : D:\android\projects

- 3) Cocher "Phone and Tablet" et spécifier les plateformes Android avec lequel l'application sera compatible.

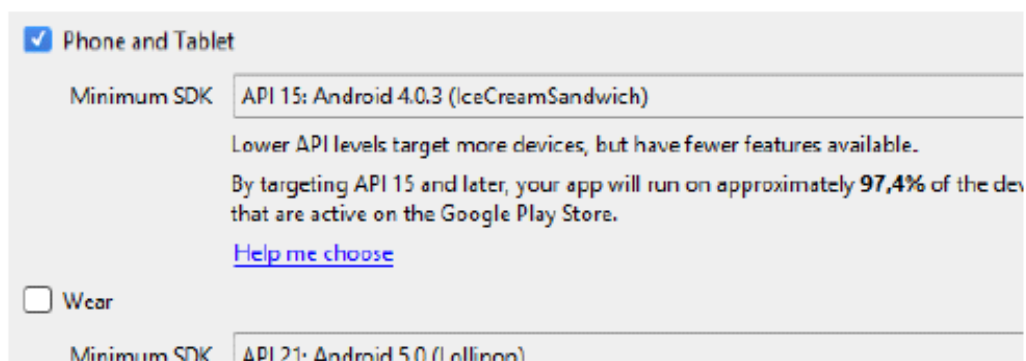


Figure 5.18 : l'affichage du résultat

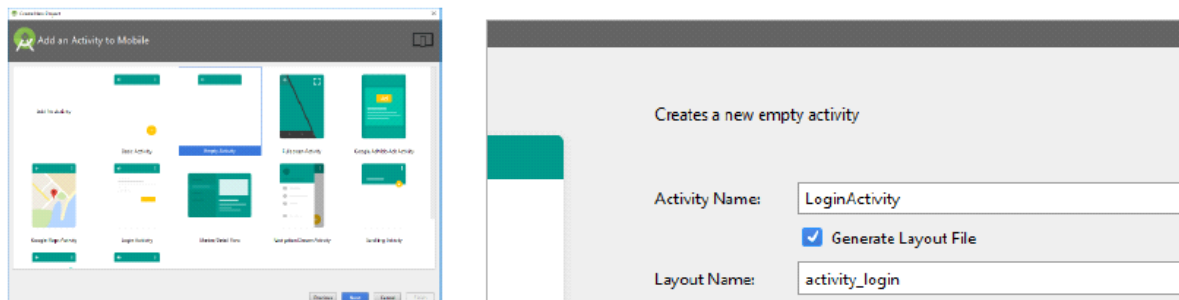
4) Android Studio propose plusieurs modèles prédéfinis, où il génèrera toutes les classes et les ressources nécessaires à la création d'un écran graphique, qui est appelé "Activité". Pour notre projet, créer un écran vide grâce à l'option "Empty Activity", car c'est mieux de commencer de zéro.

5) Enfin, il faut remplir les champs concernant le nom de l'activité comme suit :

Activity Name : LoginActivity qui est le nom de la classe qui étendra Activity. Il doit respecter la convention de nommage CamelCase (Chaque mot doit commencer par une majuscule et le reste en minuscules) suffixée par le type du contrôleur (**Activity**, **Fragment** ou **View**).

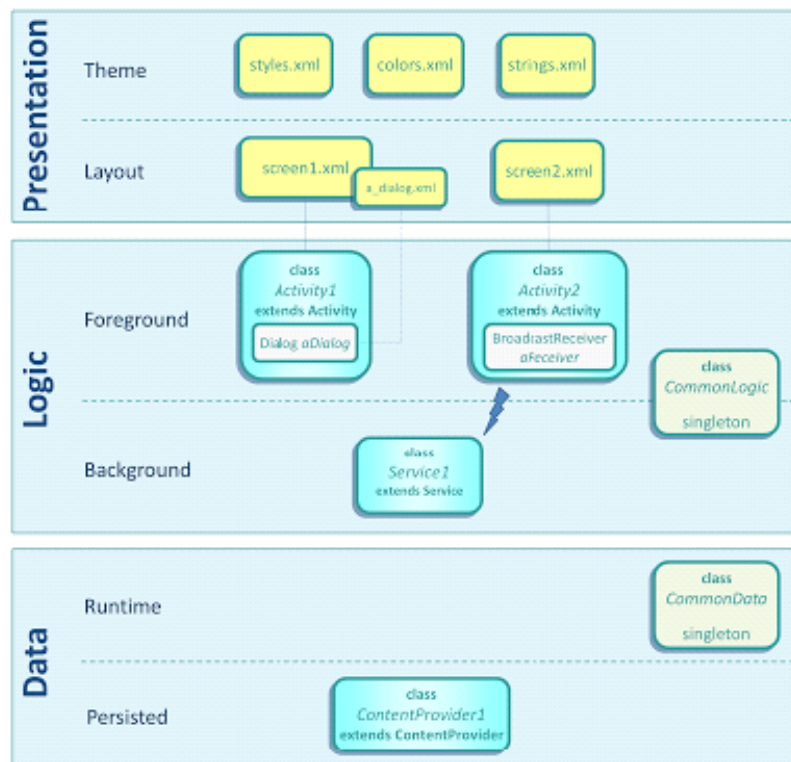
Layout Name : activity_login, qui est le nom de la vue associée à l'écran sous la forme d'un fichier XML. La convention impose que ce nom soit en snake_case (tout en minuscules et séparé avec des underscore) préfixé par le type du contrôleur.

Maintenant, le projet est créé avec une activité d'authentification vierge.



Structure d'un projet Android

Un projet Android possède une structure semblable à MVC, en séparant la vue du contrôle. Cette structure permet d'organiser le projet en 3 couches : présentation, logique et données. La figure suivante schématise cette organisation :



Un projet sous Android Studio est souvent composé de plusieurs modules, où chaque module peut être une application à part entière ou une bibliothèque. Le premier module est nommé `app` par convention, contenant toutes les sources de l'application Android organisées dans des sous-dossiers :

- `manifests` : comporte essentiellement `AndroidManifest.xml` qui décrit la configuration de l'application (titre, logo, ...).

- `java` : regroupe les sources Java de l'application organisées dans des packages (classes, interfaces, activités, ...).

- `res` : contient toutes les ressources que l'on utilise dans l'application (vues, images, couleurs, ...).

Définition d'un layout

Un layout définit la structure visuelle d'une interface graphique, d'une activité par exemple. Il est possible de déclarer un layout de 2 façons :

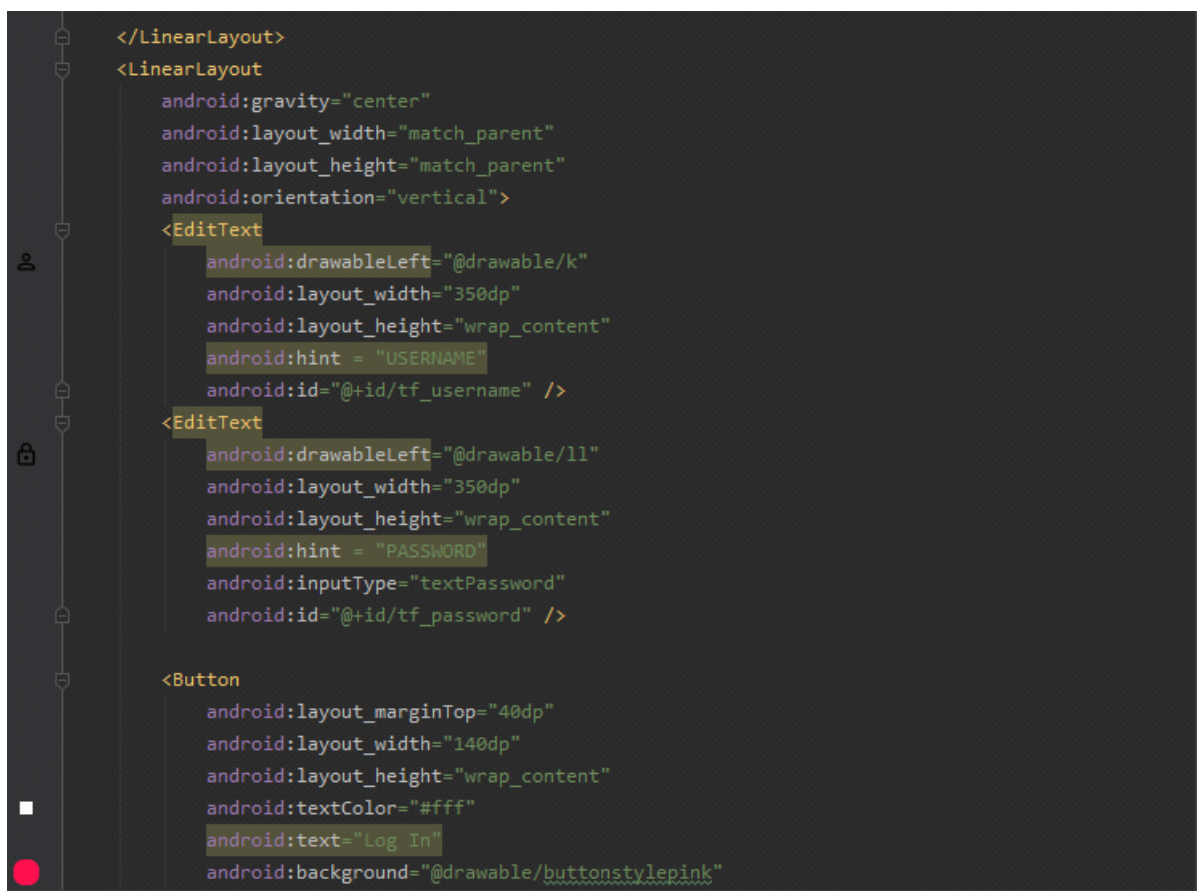
- Déclarer les éléments de l'interface graphique dans un fichier XML situé dans le dossier **res/layout/**,
- Instancier et manipuler les éléments du layout dans le code source Java au moment de l'exécution.

Déclarer un layout en XML facilite la visualisation de la structure de l'interface graphique, et ainsi résoudre les erreurs.

Voici un exemple :



La figure ci-dessous est un extrait l'interface de login pour notre application



voici le graphic :

17:14



Welcome to Web Scrapping App

 USERNAME

 PASSWORD

LOG IN



La figure ci-dessous est un extrait du layout app de dashboard de notre application :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.example.home1.MainUI">

    <LinearLayout
        android:clipToPadding="false"
        android:gravity="center"
        android:orientation="horizontal"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"

        >
        <androidx.cardview.widget.CardView
            android:layout_width="160dp"
            android:layout_height="190dp"
            android:layout_margin="10dp">

            <LinearLayout
                android:id="@+id/btn_science"
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:gravity="center"
                android:orientation="vertical">

                <ImageView
                    android:layout_width="64dp"
                    android:layout_height="64dp"
                    android:background="@drawable/cerclebackgroundpurple"
                    android:padding="10dp"
                    android:src="@drawable/ic_science" />

                <TextView
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_marginTop="10dp"
                    android:text="Science"
                    android:textStyle="bold" />

                <View
                    android:layout_width="match_parent"
                    android:layout_height="1dp"
                    android:layout_margin="10dp"
                    android:background="@color/lightgray" />

                <TextView
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:gravity="center"
                    android:padding="5dp"
                    />
            </LinearLayout>
        </CardView>
    </LinearLayout>
</LinearLayout>
```



```
</androidx.cardview.widget.CardView>
<androidx.cardview.widget.CardView
    android:layout_width="160dp"
    android:layout_height="190dp"
    android:layout_margin="10dp">

    <LinearLayout
        android:id="@+id/btn_sport"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center"
        android:orientation="vertical">

        <ImageView
            android:layout_width="64dp"
            android:layout_height="64dp"
            android:background="@drawable/cerclebackgroundpink"
            android:padding="10dp"
            android:src="@drawable/ic_sport" />

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="10dp"
            android:text="Sport"
            android:textStyle="bold" />

</LinearLayout>
</androidx.cardview.widget.CardView>
```

Layout > LinearLayout > androidx.cardview.widget.CardView > LinearLayout

IDE and Plugin Updates
Android Studio is ready to u

Voici le Dashboard de notre Application :

Articles category



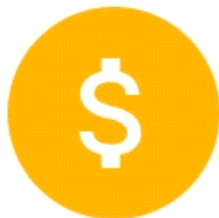
Science

Check the latest
science articles



Sport

Check the latest
Sport articles



Economy

Check the latest
economy articles



Culture

Check the latest
culture articles



Health

Maintenant que l'interface est achevée, passons à l'étape suivante "amélioration de notre code de scraping " cette dernière nous permettra de faire le display de centaines des articles au lieu de 10, et d'une part ça nous facilite les tâches pour classifier les articles selon leur catégorie avec l'intelligence artificiel dans le prochain Sprint .

création de la base de données en utilisant django

tous d'abord nous devons faire un model

Model

Un modèle est la source unique et définitive d'informations sur vos données. Il contient les champs et les comportements essentiels des données que vous stockez. En général, chaque modèle correspond à une seule table de base de données.

Les bases:

Chaque modèle est une classe Python qui sous-classe `django.db.models.Model`. Chaque attribut du modèle représente un champ de base de données. Avec tout cela, Django vous offre une API d'accès à la base de données générée automatiquement

Exemple rapide¶

Cet exemple de modèle définit une personne, qui a un prénom et un nom

```
from django.db import models

class Person(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)
```

Voici le model des articles dans notre sprint :

```
models.py
from django.db import models
from datetime import datetime
from django.contrib.auth.models import User
# Create your models here.

class Article(models.Model):

    title = models.CharField(max_length=20000)
    intro = models.CharField(max_length=20000, default="")
    link = models.CharField(max_length=1000)
    tag = models.CharField(max_length=1000)
    source = models.CharField(max_length=1000)
    date_pub = models.DateTimeField(default=datetime.now, blank=True)
    date_save = models.DateTimeField(default=datetime.now, blank=True)

    def __str__(self):
        return self.title
```

Après avoir défini les modèles, il faut indiquer à Django que vous souhaitez utiliser ces modèles. Vous pouvez le faire en éditant votre fichier de réglages et en modifiant le réglage `INSTALLED_APPS` pour y ajouter le nom du module qui contient `models.py`.

les bases de données utilisent le Sql et nous voulons transformer cette class "Article" écrit en python à un tableau dans la base de données alors en lance la commande "**manage.py makemigrations**".

un nouveau fichier apparaîtra `0001_initial.py` :

```
class Migration(migrations.Migration):

    initial = True

    dependencies = [
    ]

    operations = [
        migrations.CreateModel(
            name='Article',
            fields=[
                ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
                ('title', models.CharField(max_length=20000)),
                ('intro', models.CharField(default='', max_length=20000)),
                ('link', models.CharField(max_length=1000)),
                ('tag', models.CharField(max_length=1000)),
                ('source', models.CharField(max_length=1000)),
                ('date_pub', models.DateTimeField(blank=True, default=datetime.datetime.now)),
                ('date_save', models.DateTimeField(blank=True, default=datetime.datetime.now)),
            ],
        ),
    ]
```

puis en lance la commande "**manage.py sqlmigrate Article 0001**" et "**manage.py migrate**" (qui est responsable de l'exécution et de l'annulation des migrations)

Django Administration :

Nous avons d'abord besoin de créer un utilisateur qui peut se connecter au site d'administration. Lancez la commande suivante :

"python manage.py createsuperuser"

Saisissez le nom d'utilisateur souhaité et appuyez sur retour.

Username: admin

On vous demande alors de saisir l'adresse de courriel souhaitée :

Email address: admin@example.com

L'étape finale est de saisir le mot de passe. On vous demande de le saisir deux fois, la seconde fois étant une confirmation de la première.

Password: *****

Password (again): *****

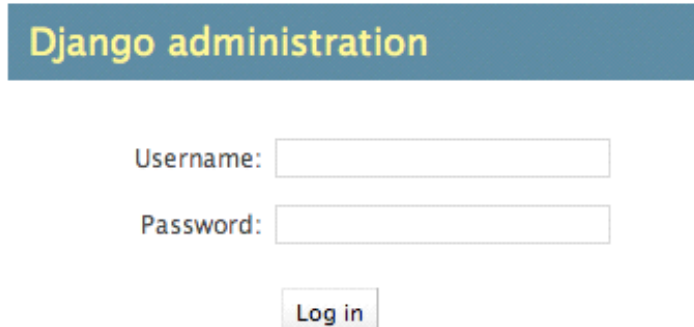
Superuser created successfully.

Démarrage du serveur de développement

Le site d'administration de Django est activé par défaut. Lançons le serveur de développement et explorons-le.

```
$ python manage.py runserver
```

À présent, ouvrez un navigateur Web et allez à l'URL « /admin/ » de votre domaine local – par exemple, <http://127.0.0.1:8000/admin/>. Vous devriez voir l'écran de connexion à l'interface d'administration :



Django administration

Username:

Password:

Rendre l'application de sondage modifiable via l'interface d'admin

Mais où est notre application de sondage ? Elle n'est pas affichée sur la page d'index de l'interface d'administration.

Juste une chose à faire : il faut indiquer à l'admin que les objets Question ont une interface d'administration. Pour ceci, ouvrez le fichier polls/admin.py et éditez-le de la manière suivante :

polls/admin.py

```

admin.py
from django.contrib import admin
from .models import Article

# Register your models here.
admin.site.register(Article)

```

Comment exécuter un script Python à partir du shell Django?

pour ce faire, créez un fichier dans un sous-répertoire de management et commands de votre app, c'est-à-dire

```

my_app/
  __init__.py
  models.py
  management/
    __init__.py
    commands/
      __init__.py
      my_command.py
  tests.py
  views.py

```

et dans ce fichier, définissez votre commande personnalisée (assurez-vous que le nom du fichier est celui de la commande à partir de ./manage.py)

```

from Django.core.management.base import BaseCommand

class Command(BaseCommand):
    def handle_noargs(self, **options):
        # now do the things that you want with your models here

```

puis intégrer le Script amélioré ("aps.py") descraping dans la sous class "commands"
ce code Scrape tous les categories et les page que on veut

L'importation des bibliothèques

```

import requests
from bs4 import BeautifulSoup

import csv
import time
import locale
import datetime
locale.setlocale(locale.LC_ALL, 'fr_FR.UTF-8')
['fr_FR.UTF-8']

from django.core.management.base import BaseCommand, CommandError
import requests
from bs4 import BeautifulSoup
from django.utils import timezone
#import pandas as pd

from test1.models import Article

def parse_date(date_txt):
    DATE_FORMAT = "%A, %d %B %Y %H:%M"
    return datetime.datetime.strptime(date_txt, DATE_FORMAT)

```

parse_data c'est une fonction qui transforme le format de la date

```

class Command(BaseCommand):
    help = 'Poling articles from APS'

    # def add_arguments(self, parser):
    #     parser.add_argument('poll_ids', nargs='+', type=int)

    def handle(self, *args, **options):
        aa="staring"
        self.stdout.write(self.style.SUCCESS('Poling articles from APS "%s"' % aa))

        categories = ["sante-science-technologie", "regions", "culture", "societe", "sport", "monde", "economie"]
        source = 'http://www.aps.dz'
        pages = 2

        for category in categories:
            tag = category
            req = requests.get(source + "/" + category)

            #soup = BeautifulSoup(req.text, 'html.parser')
            #aaa= soup.find('div', class_='k2Pagination')

            #total_pages= aaa.text[-5:].strip()

```

avec la variable "pages" on peut contrôler le nombre de pages à scraper de site Aps

```

ii =0
for page in range(1 ,pages):
    print ("----- Page: %d " %page)
    sub_req = requests.get('http://www.aps.dz/'+category+'/?start= ' +str(ii))
    sub_soup = BeautifulSoup(sub_req.text , 'html.parser')
    jj =1
    for article in sub_soup.find_all('div' ,class_='itemContainer'):
        print ("-----Article: %d " %jj)
        tag_title= article.find('h3' ,class_='catItemTitle').find("a", recursive=False)
        title =tag_title.text
        link = source+tag_title['href']
        intro = article.find('div' ,class_="catItemIntroText").text.strip()
        pub_date =article.find('span' ,class_="catItemDateCreated").text.strip()

        print("\t\tTitle: %s " %title)
        print("\t\tIntro: %s " %intro)
        print("\t\tLink : %s " %link)
        print("\t\tDate : " +str(parse_date(pub_date)))
        print("\t\tTag: %s " %tag)
        print("\t\tSource: %s " %source)

        print()
        jj =jj +1

    art = Article()
    art.title = title
    art.intro= intro
    art.link = link
    art.tag = tag
    art.source = source

```

stocker les données

voici le script pour stocker les information obtenu dans la base de données et eviter la redondance des articles :

```

num_results = Article.objects.filter(title=art.title).count()
if num_results <= 0:
    art.save()
    self.stdout.write(self.style.SUCCESS('Article saving : "%s"' % art.title))
else:
    self.stdout.write(self.style.SUCCESS('Article Already exist : "%s"' % art.title))
# ----

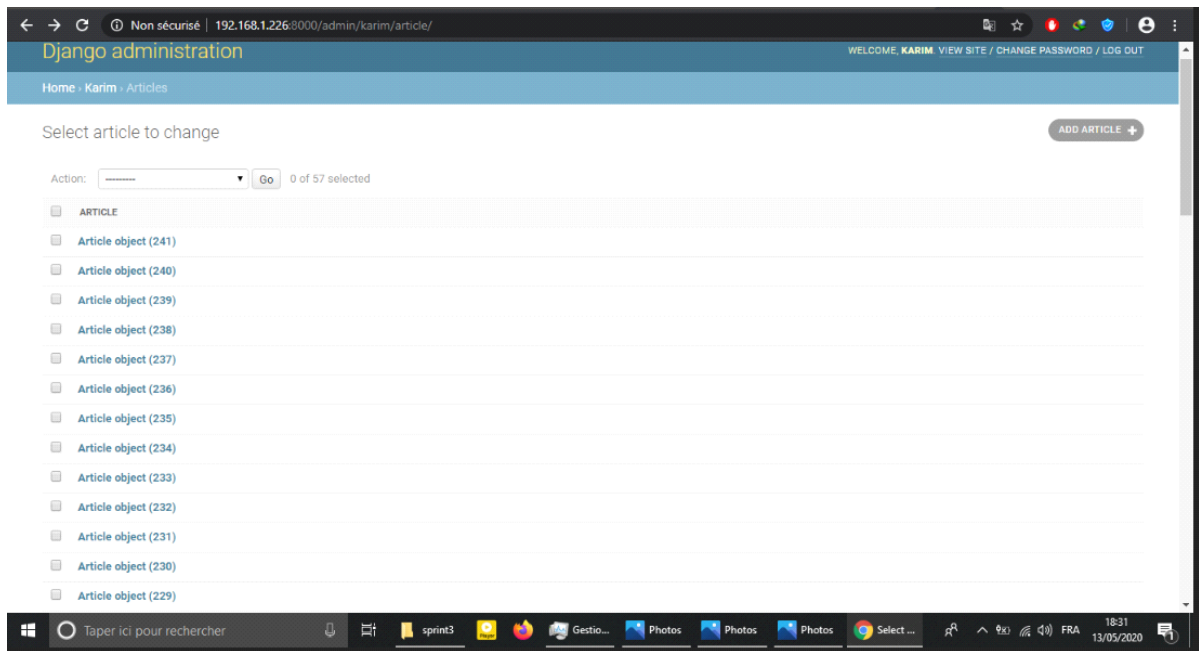
```

runscript à partir de Django-extensions :

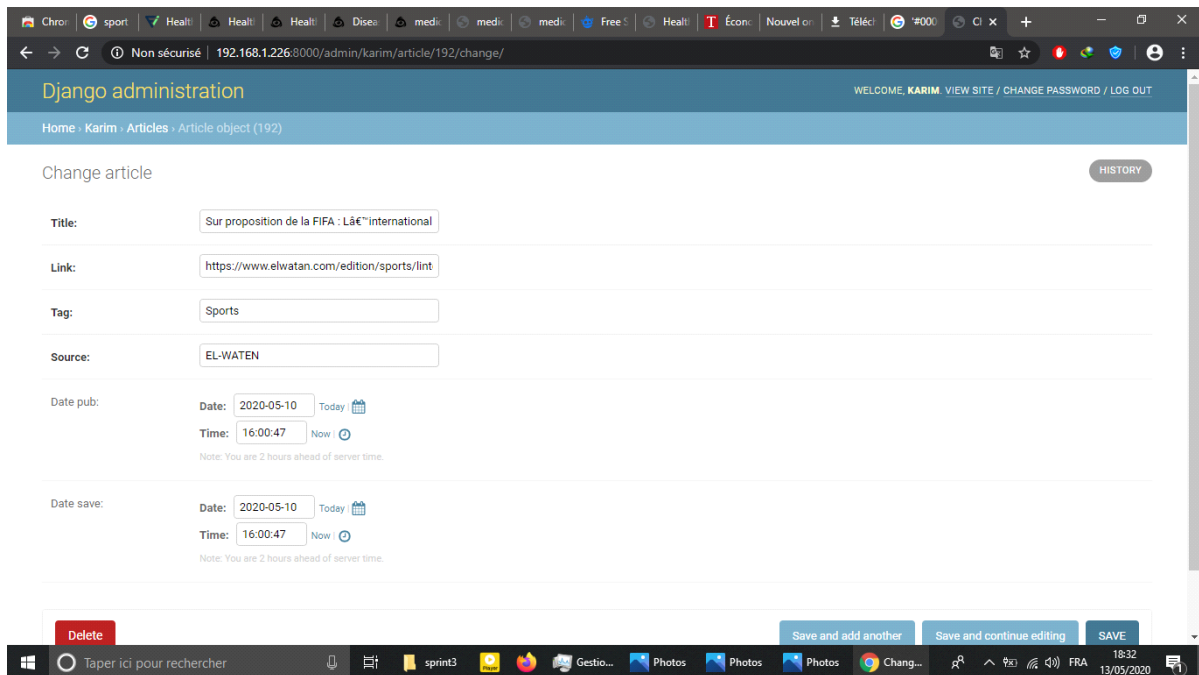
one lance la commande "python manage.py runscript aps.py"

et voici le resultat :

on lance la commande "python manage.py runserver"



on click sur un aricle pour voir l'objet :



1-Configurer Django REST Framework

D'accord, il est temps de commencer à penser à l'API de nos héros. Nous devons sérialiser les données de notre base de données via des points de terminaison. Pour ce faire, nous aurons besoin de Django REST Framework, donc installons-le. on lance la commande **"pip install djangorestframework"**

Maintenant, dites à Django que nous avons installé le Framework REST dans / settings.py:

```
INSTALLED_APPS = [  
    # All your installed apps stay the same  
    ...  
    'rest_framework',  
]
```

2-Sérialiser le modèle Article

Maintenant, nous commençons à entrer dans de nouvelles eaux. Nous devons indiquer à REST Framework notre modèle d'article et la manière dont il doit sérialiser les données.

Pour ce faire, créons un nouveau fichier - /serializers.py
Dans ce fichier, nous devons:

- 1-Importer le modèle d'article
- 2-Importer le sérialiseur de REST Framework
- 3-Créer une nouvelle classe qui relie l'article à son sérialiseur

Voici comment:

```
serializers.py > ...  
from rest_framework import serializers  
from .models import Article  
class ArticleSerializer(serializers.HyperlinkedModelSerializer):  
    class Meta:  
        model = Article  
        fields = ('title', 'link', 'tag', 'source')
```

3-Afficher les données

Il ne reste plus qu'à câbler les URL et les vues pour afficher les données!

3.1 Vues

Commençons par la vue. Nous devons rendre les différents héros au format JSON.

Pour ce faire, nous devons:

Interroger la base de données pour tous les articles

Passez cet ensemble de requêtes de base de données dans le sérialiseur que nous venons de créer, afin qu'il soit converti en JSON et rendu

Dans myapi / views.py:

```
# Create your views here.
def home(request):

    return HttpResponse('<h1> karim site </h1>')

class ArticleViewSet(APIView):

    serializer_class = ArticleSerializer

    parser_classes = [JSONParser]

    def post(self, request, ):

        queryset = Article.objects.all()
        serializer = ArticleSerializer(queryset, many=True)
        return Response( serializer.data)

        #return Response({'received data': request.data})
    def post1(self):
        pass
```

3.2-Site URLs

La dernière étape consiste à pointer une URL sur l'ensemble de vues que nous venons de créer.

Dans Django, les URL sont d'abord résolues au niveau du projet. Il y a donc un fichier dans le répertoire mysite appelé urls.py.

Allez là-bas. Vous verrez que l'URL du site d'administration s'y trouve déjà. Il ne nous reste plus qu'à ajouter une URL pour notre API. Pour l'instant, mettons simplement notre API à l'index:

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('myapi.urls')),
    path('api-token-auth/', obtain_auth_token, name='api_token_auth')
]
```

Testez-le!

Redémarrez le serveur Django:

```
$ python manage.py runserver
```

Article List

OPTIONS

GET ▾

GET /karim/articles/?format=api

HTTP 200 OK

Allow: GET, POST, HEAD, OPTIONS

Content-Type: application/json

Vary: Accept

```
{
  {
    "id": 236,
    "title": "75e anniversaire des massacres du 8 mai 1945 : Le 7e RTA et le supplic...",
    "link": "https://www.elwatan.com/pages-hebdo/histoire/___trashed-10-05-2020",
    "tag": "Histoire",
    "source": "EL-WATEN"
  },
  {
    "id": 210,
    "title": "A\u00e2n T\u00e0mouchent : Retour aux restrictions de pr\u00e9vention contre le Co...",
    "link": "https://www.elwatan.com/regions/ouest/actu-ouest/temouchent-retour-aux-restrictions-de-prevention-contre-le-covid-19-09-05-2020",
    "tag": "Ouest",
    "source": "EL-WATEN"
  },
  {
    "id": 225,
    "title": "Ahcene Zehnati . \u00c9conomiste de la sant\u00e9 et chercheur au CREAD : \u00c0Il...",
    "link": "https://www.elwatan.com/pages-hebdo/sup-eco/ahcene-zehnati-economiste-de-la-sante-et-chercheur-au-cread-il-est-indispensable-de-disposer-dun-systeme-dinfor",
    "tag": "Sup-Eco",
    "source": "EL-WATEN"
  }
}
```